Finding Time Complexity for Recurrences:

Identify the number of times a recursive call____________and what the new ____________ is

Local time complexity is____

Note:

- c is generally used as a ____________
- c _____ Θ(1)
- n _____ cn _____ Θ(n)

```
int foo(int N){
  int a,b,c;
  if(N<=3) return 1500; // Note N<=3
  a = 2*foo(N-1);
 // a = foo(N-1)+foo(N-1);
  printf("A");
  b = foo(N/2);
  c = foo(N-1);
  return a+b+c;
}
```

Base case: T(__) = __________

Recursive case: T(__) = _____________________

T(N) gives us the Time Complexity for foo(N). We need to solve it (find the closed form)

```
void bar(int N){
 int i,k,t;
 if(N<=1) return;
 bar(N/5);
 for(i=1;i<=5;i++){
   bar(N/5);
 }
 for(i=1;i<=N;i++){
   for(k=N;k>=1;k--)
     for(t=2;t<2*N;t=t+2)
       printf("B");
 }
 bar(N/5);
}
```

Base case: T(__) = __________

Recursive case: T(__) = _____________________

Solve T(N)

Let N be the number of elements to process in this call. N = right-left+1

```
int binary_search(int A[], int left, int right, int v){
   int m = left+(right-left)/2;
   if (left > right) return -1;
   if (v == A[m]) return m;
   if (v < A[m])
      return binary_search(A, left, m-1, v);
   else
      return binary_search(A, m+1, right, v);
}
```

Recurrence: base case: T(   ) = T (   ) = ________

recursive case: T(   ) = _______________

Draw TC tree. Use it to find TC. TC = __________________

SC = ____________

```
Merge_sort(A, le, r)   //N = ri-le+1
   if (le>=ri) return
   else
      m = floor(le+(ri-le)/2)
      Merge sort(A, le,  m);
      Merge_sort(A, m+1, ri);
      Merge(A, le, m, ri);
```

Recurrence: base case: T(   ) = T (   ) = ________

recursive case: T(   ) = _______________

```
Merge(A, le, m, ri)
   n1=m-le+1+1    // +1 for inf
   n2=ri-m+1      // +1 for inf
   let L[n1], R[n2] be arrays
   for j=0 to n1-2
      L[j]=A[le+j]
   for j=0 to n2-2
      R[j]=A[m+1+j]
   L[n1] = inf
   R[n2] = inf
   j=0,
   i=0
   for k=le to ri
      if L[i] ≤ R[i]
         A[k]=L[i]
         i++
      else
         A[k] = R[j]
         j++
// pseudocode
// - indentation => instruction group in {}
// - loops:   for k=le to ri    means  for(k=le; k<=ri; k++)
```